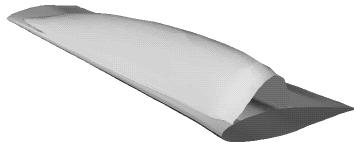


CUMULVS

Tutorial

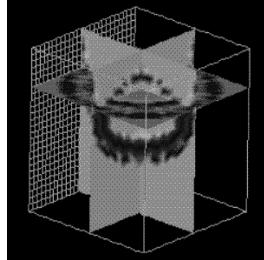


ACTS Toolkit Workshop

Dr. James Arthur Kohl

Computer Science and Mathematics Division

Oak Ridge National Laboratory



September 30, 2000

Research sponsored by the Applied Mathematical Sciences Research Program, Office of Mathematical, Information, and Computer Sciences, U.S. Department of Energy, under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

Kohl/2000-1



Scientific Simulation Issues...

- Fundamental Parallel Programming
 - Synchronization, Coordination and *Control*
- Distributed Data Organization
 - Locality, Latency Hiding, *Data Movement*
- Long-Running Simulation Experiments
 - *Monitoring, Fault Recovery*
- Massive Amounts of Data / Information
 - Archival Storage, *Visualization*
- Too Much Computer, Not Enough Science!
 - *Need Some Help...*

ORNL

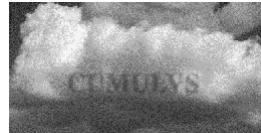
Kohl/2000-2

Potential Benefits from Computer Science Infrastructure:

- **On-The-Fly Visualization**
 - ⇒ Interactive Access to Intermediate Results
 - ⇒ Attached as Needed, Minimize Overhead
- **Computational Steering**
 - ⇒ Apply Visual Feedback to Alter Course / Restart
 - ⇒ “Close Loop” on Experimentation Cycle
- **Fault Tolerance**
 - ⇒ Automatic Fault Recovery / Load Balancing
 - ⇒ Keep Long-Running Simulations Running Long

ORNL

Kohl/2000-3



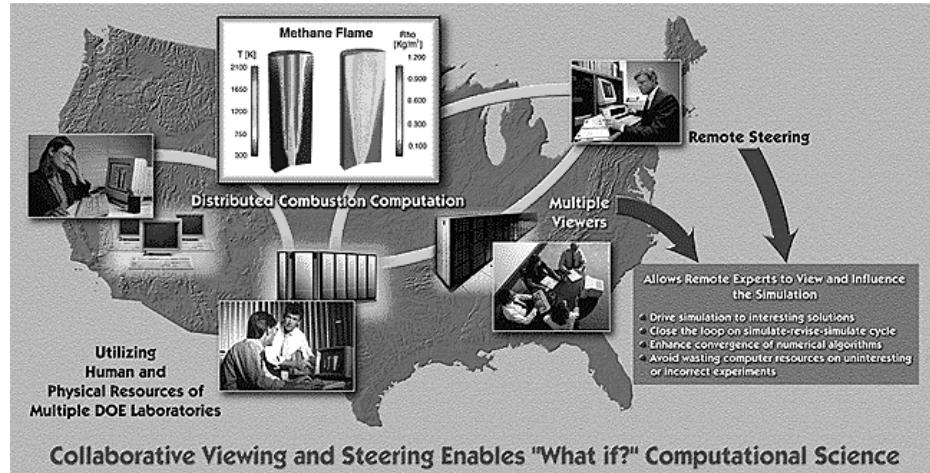
(Collaborative, User Migration, User Library for Visualization and Steering)

- Collaborative Infrastructure for Interacting with Scientific Simulations:
 - ⇒ Run-Time Visualization by Multiple Viewers
 - Dynamic Attachment, Independent Views
 - ⇒ Coordinated Computational Steering
 - Model & Algorithm
 - ⇒ Heterogeneous Checkpointing / Fault Tolerance
 - Automatic Fault Recovery and Task Migration
 - ⇒ Coupled Models...

ORNL

Kohl/2000-4

Collaborative Combustion Simulation



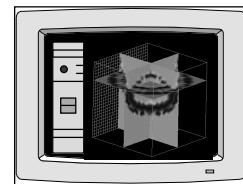
ORNL

Kohl/2000-5

CUMULVS Visualization Features

⇒ Interactive Visualization

- * Simple API for Scientific Visualization
- * Use Your Favorite Visualization Tool

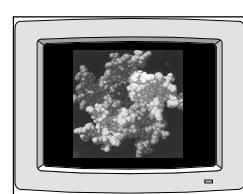


⇒ Minimize Overhead When No Viewers

- * One Message Probe, No Application Penalty

⇒ Send Only Viewed Data

- * Partial Array / Lower Resolution



⇒ Rect Mesh & Particle Data

⇒ HPF-type Data Distributions

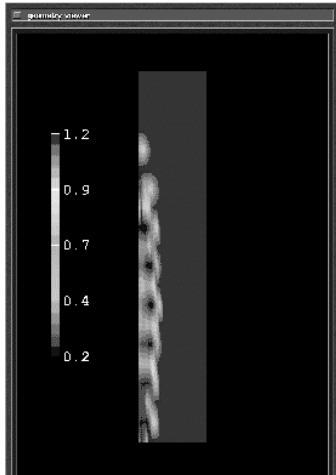
- * BLOCK, CYCLIC, EXPLICIT, COLLAPSE

⇒ Soon Unstructured & Adaptive Meshes...

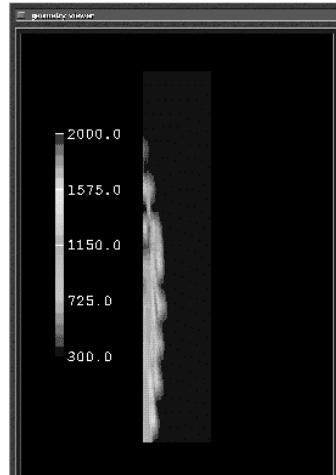
ORNL

Kohl/2000-6

Multiple Simultaneous Views



Density

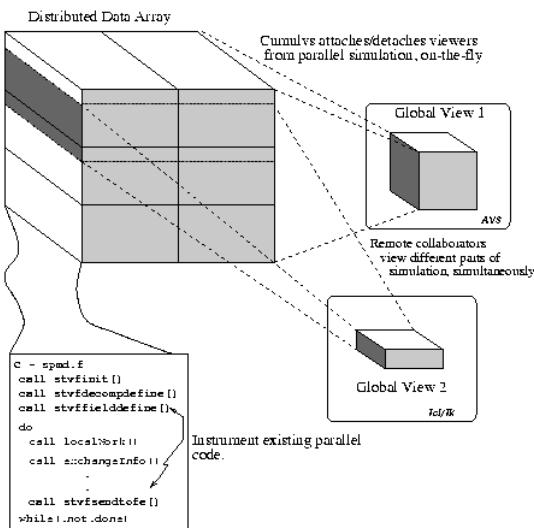


Temperature

ORNL

Kohl/2000-7

Multiple Distinct Views

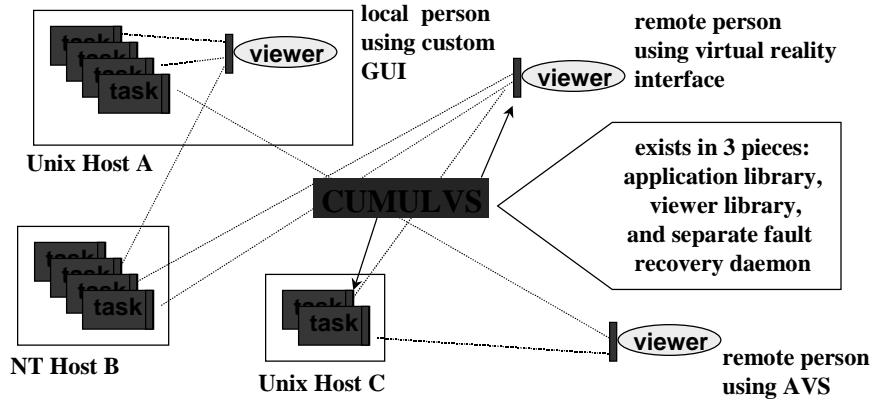


ORNL

Kohl/2000-8

CUMULVS Architecture

coordinate the consistent collection and dissemination of information to / from parallel tasks to multiple viewers



interact with distributed / parallel application or simulation
supports most target platforms (PVM / MPI, Unix / NT, etc.)

Kohl/2000-9

CUMULVS Environment

- Set STV_ROOT Environment Variable:
or `setenv STV_ROOT /home/user/CUMULVS`
`or export STV_ROOT=/home/user/CUMULVS`
- Use Sample “Makefile.aimk” (in examples/src/):

```
STV_VIS = NONE
STV_MP = PVM
include $(STV_ROOT)/src/Makeincl.stv
→ $(STV_CFLAGS), $(STV_LDFLAGS), $(STV_LIBS)...
```
- Include CUMULVS Header Files (in include/):
 ⇒ Defines Constants, Functions, Types

C:	<code>#include <stv.h></code>
Fortran:	<code>include 'fstv.h'</code>

ORNL

Kohl/2000-10

Instrumenting Simulations Or Other Applications

- CUMULVS System Initialization ~ One Call
 - ⇒ Each Task: Logical Application Name, # of Tasks
- Data Fields (Visualization & Checkpointing)
 - ⇒ Data Distribution: Dim, Decomp, PE Topology
 - ⇒ Local Allocation: Name, Type, Size, Offsets
- Steering Parameters
 - ⇒ Logical Name, Data Type, Reference to Storage
- Typically 10s of Lines of Code...

ORNL

Kohl/2000-11

CUMULVS Initialization

- C: `stv_init(char *appname, int msgtag,
int ntasks, int nodeid);`
- Fortran: `stvinit(appname, msgtag, ntasks, nodeid)`
where:
 - appname ~ logical name of application
 - msgtag ~ message tag to reserve for CUMULVS*
 - ntasks ~ number of tasks in application
 - nodeid ~ task index of the caller

E.g.: `stv_init("solver", 123, 32, 3);`

* For Backwards Compatibility with PVM 3.3 (Before Message Contexts...).

ORNL

Kohl/2000-12

Distributed Data Decomposition

- C: `int decompId = stv_decompDefine(int dataDim,
int *axisType, int *axisInfo1, int *axisInfo2,
int *glb, int *gub, int prank, int *pshape);`
- Fortran: `stvfdecompdefine(dataDim, axisType, axisInfo1,
axisInfo2, glb, gub, prank, pshape, decompId)`

where:

- decompId ~ integer decomp handle returned by CUMULVS
- dataDim ~ dimensionality of data decomposition
- axisType ~ decomp type identifier, for each axis
 - * `stvBlock`, `stvCyclic`, `stvExplicit`, `stvCollapse`
- axisInfo1,2 ~ specific decomposition details, per axis
 - * E.g. axisInfo1 = Block Size or Cycle Size
 - * E.g. axisInfo1 = Explicit Lower Bound, axisInfo2 = Upper Bound
 - * Note: axisInfo1,2 can be set to `stvDefaultInfo`

ORNL

Kohl/2000-13

Data Decomposition (continued)

Global Bounds and Pshape

- C: `stv_decompDefine(...,
int *glb, int *gub, int prank, int *pshape);`
- Fortran: `stvfdecompdefine(..., glb, gub,
prank, pshape)`

where:

- glb ~ lower bounds of decomp in global coordinates
- gub ~ upper bounds of decomp in global coordinates
- prank ~ dimensionality of processor topology (pshape)
- pshape ~ logical processor organization / topology
 - * number of processors per axis

ORNL

Kohl/2000-14

Example Decomposition

3D Standard Block Decomp on 2D Processor Array

```
int decompId, axisType[3], glb[3], gub[3], pshape[2];

axisType[0] = axisType[1] = stvBlock;
axisType[2] = stvCollapse;

glb[0] = glb[1] = glb[2] = 1;
gub[0] = gub[1] = gub[2] = 100;

pshape[0] = 4;      /* "Standard" Block Size = 25 */
pshape[1] = 5;      /* "Standard" Block Size = 20 */

decompId = stv_decompDefine( 3,
    axisType, stvDefaultInfo, stvDefaultInfo,
    glb, gub, 2, pshape );
```

ORNL

Kohl/2000-15

Example Decomposition

3D Block-Cyclic Decomp on 2D Processor Array

```
int axisType[3], axisInfo[3], glb[3], gub[3], pshape[2];

axisType[0] = stvBlock;  axisInfo[0] = 10; /* Block */
axisType[1] = stvCyclic; axisInfo[1] = 3;  /* Cycle */
axisType[2] = stvCollapse;

glb[0] = glb[1] = glb[2] = 1;
gub[0] = 100;  gub[1] = 150;  gub[2] = 200;

pshape[0] = 5;  pshape[1] = 5;

decompId = stv_decompDefine( 3, axisType,
    axisInfo, stvDefaultInfo, glb, gub, 2, pshape );
```

ORNL

Kohl/2000-16

Example Decomposition

3D Explicit Decomp on 3D Processor Array

```
int axisType[3], axisInfo1[3], axisInfo2[3],... pshape[3];

axisType[0] = axisType[1] = axisType[2] = stvExplicit;
axisInfo1[0] = 11;  axisInfo2[0] = 17;
axisInfo1[1] = 33;  axisInfo2[1] = 57;
axisInfo1[2] = 1;   axisInfo2[2] = 7;

glb[0] = glb[1] = glb[2] = 1;
gub[0] = 80;  gub[1] = 90;  gub[2] = 20;

pshape[0] = 3;  pshape[1] = 5;  pshape[2] = 2;

decompId = stv_decompDefine( 3, axisType,
                           axisInfo1, axisInfo2, glb, gub, 3, pshape );

```

ORNL

Kohl/2000-17

Additional Explicit Bounds

For Extra Explicit Array Subregions Per Task

- C: `int stv_decompAddExplicitBounds(int decompId,
 int axis, int lowerBound, int upperBound);`
- Fortran: `stvfdecompadexplicitbounds(decompId,
 axis, lowerBound, upperBound, info)`

where:

- decompId ~ integer handle to CUMULVS decomp
- axis ~ index of desired decomposition axis
- lowerBound ~ lower bound of subregion (global coords)
- upperBound ~ upper bound of subregion (global coords)

ORNL

Kohl/2000-18

Additional Explicit Bounds – Vector Version For Multiple Additional Explicit Array Subregions

- C:

```
int stv_decompAddExplicitVBounds( int decompId,
                                      int axis, int *lowerBounds, int *upperBounds,
                                      int nbounds );
```
 - Fortran:

```
stvfdecompaddexplicitvbounds( decompId,
                                         axis, lowerBounds, upperBounds, nbounds, info )
```
- where:
- lowerBounds ~ array of lower bounds (global coords)
 - upperBounds ~ array of upper bounds (global coords)

ORNL

Kohl/2000-19

Example Decomposition 3D Local Array

```
int decompId, glb[3], gub[3];

glb[0] = glb[1] = glb[2] = 0;
gub[0] = gub[1] = gub[2] = 100;

decompId = stv_decompDefine( 3,
                            stvLocalArray, stvLocalArray, stvLocalArray,
                            glb, gub, 1, stvLocalArray );
```

ORNL

Kohl/2000-20

Data Field Definition

- C: `int fieldId = stv_fieldDefine(STV_VALUE var,
char *name, int decompId, int *arrayOffset,
int *arrayDecl, int type, int *paddr, int aflag);`
 - Fortran: `stvffielddefine(var, name, decompId,
arrayOffset, arrayDecl, type, paddr, aflag, fieldId)`
- where:
- fieldId ~ integer field handle returned by CUMULVS
 - var ~ reference (pointer) to local data array storage
 - name ~ logical name of data field
 - decompId ~ handle to pre-defined decomposition template
 - type ~ integer constant that defines data type
 - * `stvByte, stvInt, stvFloat, stvDouble, stvLong, stvCplx, stvUint, ...`

ORNL

Kohl/2000-21

Data Field Definition

Local Data Field Allocation

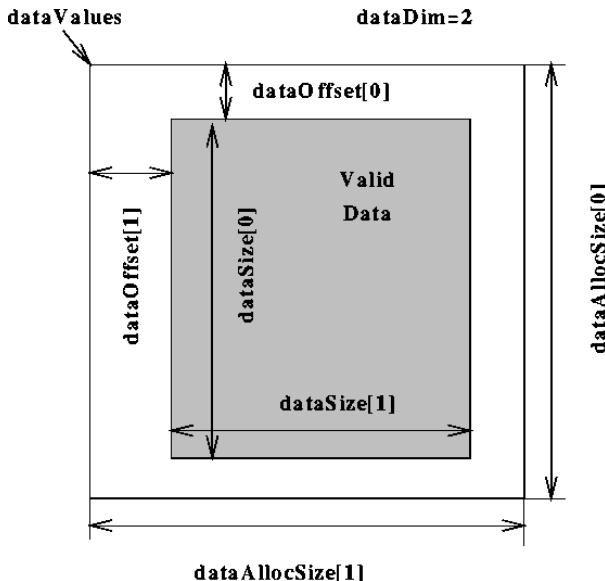
- C: `int fieldId = stv_fieldDefine(...,
int *arrayOffset, int *arrayDecl, ...);`
 - Fortran: `stvffielddefine(..., arrayOffset, arrayDecl ...)`
- where:
- arrayOffset ~ offset to “valid” data, along each axis
 - arrayDecl ~ overall allocated array size, along each axis

Note: Size of actual “valid” data is determined in conjunction
with data decomposition information...

ORNL

Kohl/2000-22

Local Allocation Organization



ORNL

Kohl/2000-23

Data Field Definition

Processor “Address”

- C: `int fieldId = stv_fieldDefine(..., int *paddr, ...);`
- Fortran: `stvffielddefine(..., paddr, ...)`

where:

→ paddr ~ integer array containing the local task’s position
in the overall processor topology (pshape)
* paddr index starts from 0!
(NOT the “physical” processor address!)

E.g. for Pshape = { 4, 4, 2 } and Prank = 3, Paddr = { 2, 3, 1 } is:
the 3rd processor of 4 along axis 1
the 4th processor of 4 along axis 2
the 2nd processor of 2 along axis 3

ORNL

Kohl/2000-24

Data Field Definition

Access / Usage Flag

- C: `int fieldId = stv_fieldDefine(..., int aflag);`
- Fortran: `stvffielddefine(..., aflag, ...)`

where:

- `aflag` ~ integer flag to indicate external access to field:
 - * `stvVisOnly` ~ access data field for visualization only
 - * `stvCpOnly` ~ access data field for checkpointing only
 - * `stvVisCp` ~ access data field for visualization *and* checkpointing

ORNL

Kohl/2000-25

Example Data Field

100x100x100 Float Array on Processor { 1, 2 } of Decomp
(E.g., say, as part of 1000x1000x1000 Global Decomp)

```
float foo[100][100][100];
int fieldId, declare[3], paddr[2];

declare[0] = declare[1] = declare[2] = 100;

paddr[0] = 1;  paddr[1] = 2;

fieldId = stv_fieldDefine( foo, "Foo", decompId,
                           stvNoFieldOffsets, declare, stvFloat, paddr,
                           stvVisOnly );
```

ORNL

Kohl/2000-26

Example Data Field

80x80x60 Double Array on Processor { 2, 1, 3 } of Decomp
(With 10 Element Neighbor Boundary in Local Allocation)

```
double bar[100][100][80];
int fieldId, offsets[3], declare[3], paddr[3];

offsets[0] = offsets[1] = offsets[2] = 10;

declare[0] = declare[1] = 100; declare[2] = 80;

paddr[0] = 2; paddr[1] = 1; paddr[2] = 3;

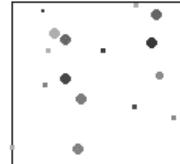
fieldId = stv_fieldDefine( bar, "Bar", decompId,
                           offsets, declare, stvDouble, paddr, stvVisCp );
```

ORNL

Kohl/2000-27

CUMULVS Particle Handling

- Particle Data Fundamentally Different
 - ⇒ Data Fields Encapsulated in a Particle Container
 - ⇒ Explicit Coordinates Per Particle
- Particle-Based Decomposition API
 - ⇒ User-Defined, Vectored Accessor Routines
- Viewing Particle Data
 - ⇒ AVS Module Extensions
 - ⇒ Tcl/Tk Slicer Particle Mode



ORNL

Kohl/2000-28

Particle Definition

A Container for Particle Data Fields

- C: `int particleId = stv_particleDefine(char *name,
int dataDim, int *glb, int *gub, int ntasks,
STV_GET_PARTICLE get_particle,
STV_MAKE_PARTICLE make_particle);`
- Fortran: `stvfparticledefine(name, dataDim, glb, gub,
ntasks, get_particle, make_particle, particleId)`

where:

- particleId ~ integer particle handle returned by CUMULVS
- name ~ logical name of particle wrapper
- dataDim ~ dimensionality of particle space
- glb, gub ~ global bounds of particle space
- ntasks ~ number of parallel tasks cooperating on particles

ORNL

Kohl/2000-29

Particle Definition

User-Defined Particle Functions

- C: `int particleId = stv_particleDefine(...,
STV_GET_PARTICLE get_particle,
STV_MAKE_PARTICLE make_particle);`
- Fortran: `stvfparticledefine(..., get_particle,
make_particle, ...)`

where:

- `get_particle()` ~ user-defined accessor function
 - * returns a user-defined handle to CUMULVS for the given particle
- `make_particle()` ~ user-defined constructor function
 - * allow CUMULVS to create a new particle (checkpoint recovery)
 - (Can set `make_particle()` to NULL if not checkpointing)

ORNL

Kohl/2000-30

Get a Particle for CUMULVS

Particle Accessor Function

- C: `void get_particle(int index, STV_REGION region,
STV_PARTICLE_ID *id, int *coords);`
- Fortran: `get_particle(index, region, id, coords)`

where:

- index ~ requested particle index
 - * positive integer value, index set to **-1** for search reset
 - * if particle not found for given index, should return **id** as NULL
- region ~ subregion of particle space to search
 - * can use stv_particle_in_region() utility function...
- id ~ user-defined particle handle returned by search
 - * (void *) → anything the application wants to cast it to...
- coords ~ array of coordinates for returned particle

ORNL

Kohl/2000-31

Make a Particle for CUMULVS

Particle Constructor Function – Checkpoint Recovery

- C: `void make_particle(int *coords,
STV_PARTICLE_ID *id);`
- Fortran: `make_particle(coords, id)`

where:

- coords ~ array of coordinates for requested particle creation
- id ~ user-defined particle handle returned for new particle
 - * (void *) → anything the application wants to cast it to...

ORNL

Kohl/2000-32

Particle Field Definition

Actual Data Fields Associated with a Particle

- C:

```
int pfieldId = stv_pfieldDefine( char *name,
    int particleId, int type, int nelems,
    STV_GET_PFIELD get_pfield, STV_VALUE get_pfield_arg,
    STV_SET_PFIELD set_pfield, STV_VALUE set_pfield_arg,
    int aflag );
```
- Fortran:

```
stvfpfielddefine( name, particleId, type, nelems,
    get_pfield, get_pfield_arg, set_pfield, set_pfield_arg,
    aflag, pfieldId )
```

where:

- pfieldId ~ returned integer particle field handle
- name ~ logical name of particle data field
- particleId ~ handle to encapsulating particle wrapper

ORNL

Kohl/2000-33

Particle Field Definition (cont.)

Actual Data Fields Associated with a Particle

- C:

```
stv_pfieldDefine( ..., int type, int nelems, ...,
    int aflag );
```
- Fortran:

```
stvfpfielddefine( ..., type, nelems, ...,
    aflag, ... )
```

where:

- type ~ data type for particle field (**stvInt**, **stvFloat**, etc.)
- nelems ~ number of data elements
 - * only simple 1-D arrays (vectors) are supported as particle fields
- aflag ~ same as for `stv_fieldDefine()`, access flag
 - * **stvVisOnly**, **stvCpOnly**, or **stvVisCp**.

ORNL

Kohl/2000-34

Particle Field Definition (cont.)

User-Defined Particle Field Functions

- C: `stv_pfieldDefine(...,`
 `STV_GET_PFIELD get_pfield, STV_VALUE get_pfield_arg,`
 `STV_SET_PFIELD set_pfield, STV_VALUE set_pfield_arg ...);`
- Fortran: `stvfpfielddefine(...,`
 `get_pfield, get_pfield_arg,`
 `set_pfield, set_pfield_arg, ...)`

where:

- `get_pfield()` / `get_pfield_arg` ~ user-defined accessor / arg
 - * return reference to actual data storage
 - * extra user-defined argument, (`void *`) for vectored `get_pfield()` impl
- `set_pfield()` / `set_pfield_arg` ~ user-defined constructor / arg
 - * for checkpoint recovery, set the values of given particle field array
 - * extra user-defined argument, (`void *`) for vectored `set_pfield()` impl

ORNL

Kohl/2000-35

Get a Particle Field for CUMULVS

Particle Field Accessor Function

- C: `void get_pfield(STV_PARTICLE_ID id,`
 `STV_VALUE user_data, STV_VALUE *data);`
- Fortran: `get_pfield(id, user_data, data);`

where:

- `id` ~ user-defined particle handle
 - * as returned by previous `get_particle()` call
- `user_data` ~ arbitrary user-defined argument
 - * (`void *`) value for vectored implementation of `get_pfield()`
- `data` ~ reference to actual data storage

E.g. `if (user_data == "density")`
 `*data = ((mystruct) id)->density;`

ORNL

Kohl/2000-36

Set a Particle Field for CUMULVS

Particle Field Constructor Function – Checkpoint Recovery

- C: `void set_pfield(STV_PARTICLE_ID id,
STV_VALUE user_data, STV_VALUE data);`
- Fortran: `set_pfield(id, user_data, data);`

where:

- id ~ user-defined particle handle
 - * as returned by previous `get_particle()` call
- user_data ~ arbitrary user-defined argument
 - * (void *) value for vectored implementation of `set_pfield()`
- data ~ reference to new data values to set

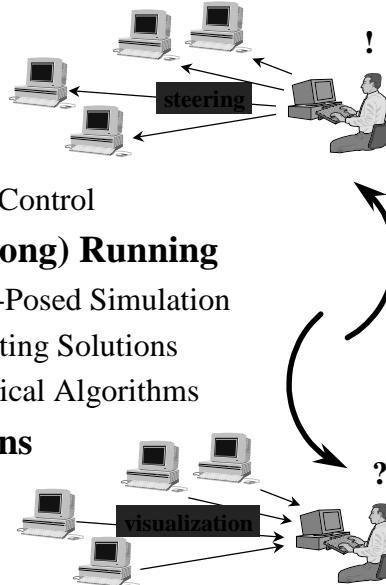
E.g. `if (user_data == "density")
((mystruct) id)->density = data;`

ORNL

Kohl/2000-37

CUMULVS Steering Features

- Computational Steering
 - ⇒ API for Interactive Application Control
- Modify Parameters While (Long) Running
 - * Eliminate Wasteful Cycles of Ill-Posed Simulation
 - * Drive Simulation to More Interesting Solutions
 - * Enhance Convergence of Numerical Algorithms
- Allows “What If” Explorations
 - * Closes Loop of Standard Simulation Cycle
 - * Explore Non-Physical Effects...

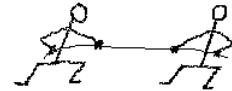


ORNL

Kohl/2000-38

Coordinated Steering

- Multiple, Remote Collaborators
- Simultaneously Steer Different Parameters
 - ⇒ Physical Parameters of Simulation
 - ⇒ Algorithmic Parameters ~ e.g. Convergence Rate
- Cooperate with Collaborators
 - ⇒ Parameter Locking ~ Prevent Conflicts
 - ⇒ Vectored Parameters...
- Parallel / Distributed Simulations
 - ⇒ Synchronize with Parallel Tasks
 - ⇒ All Tasks Update Parameter in Unison



ORNL

Kohl/2000-39

Steering Parameter Definition

- C:

```
int paramId = stv_paramDefine( char *name,
                                STV_VALUE var, int type, int aflag );
```
 - Fortran: `stvfparamdefine(name, var, type, aflag)`
- where:
- paramId ~ returned integer steering parameter handle
 - name ~ logical name of steering parameter
 - var ~ reference to actual steering parameter storage
 - type ~ data type of steering parameter (**stvInt**, **stvFloat**, ...)
 - aflag ~ external access flag, same as before
 - * **stvVisOnly**, **stvCpOnly** or **stvVisCp**

ORNL

Kohl/2000-40

Steering Parameter Definition

Vector Parameters

- C:

```
int paramId = stv_vparamDefine( char *name,
    STV_VALUE *vars, char **pnames, int *types, int num,
    int aflag );
```
- Fortran: (no Fortran equivalent yet)

where:

- paramId ~ returned integer steering vector handle
- name ~ logical name of steering vector
- vars ~ array of references to steering element storage
- pnames ~ array of logical names for steering elements
- types ~ array of steering element data types
 - * OR with **stvIndex** for “indexed” steering vector (**stvInt** | **stvIndex**)
- aflag ~ external access flag (**stvVisOnly**, **stvCpOnly**, **stvVisCp**)

ORNL

Kohl/2000-41

Steering Parameter Updates...

- C:

```
int changed = stv_isParamChanged( int paramId );
```
- Fortran: **stvfisparamchanged(paramId)**

where:

- paramId ~ integer steering parameter handle
- changed ~ status of steering parameter
 - * > 0 indicates an update has occurred
 - * = 0 indicates no change to parameter value
 - * < 0 indicates an error condition (bad parameter id, etc)

ORNL

Kohl/2000-42

Passing Control to CUMULVS

- After Data Fields and Parameters Are Defined
 - ⇒ Single Periodic Call to Pass Control to CUMULVS
 - ⇒ Once Per Iteration? Between Compute Phases?

C: `int params_changed = stv_sendReadyData(int update_field_times);`

Fortran: `stvfsendreadydata(update_field_times)`

where:

→ `params_changed` ~ return code that indicates whether any steering parameter values have been updated

* > 0 an update has occurred, = 0 no changes to any values

→ `update_field_times` ~ flag to control data field times

* in general, just use `stvSendDefault...` ☺

ORNL

Kohl/2000-43

CUMULVS Fault Tolerance Features

- **Application Fault Tolerance**
 - ⇒ Automatic Detection and Recovery from Failures
- **User Directed Checkpointing**
 - ⇒ User Decides What / Where to Checkpoint
 - ⇒ Minimizes Amount of Stored Data
- **Heterogeneous Task Migration**
 - ⇒ Restart Tasks on Heterogeneous Hosts
 - ⇒ Restart is Automatically *Repartitioned* if Host Pool is of Different Size or Topology (Yikes!)
- **Avoids Synchronizing Distributed Tasks**
 - ⇒ Asynchronous Checkpoint Collection and Fault Detection
 - ⇒ Minimize Intrusion of Checkpoint / Restart

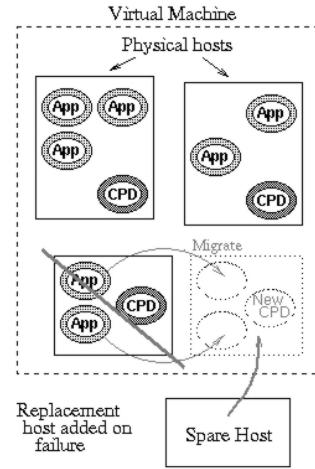


ORNL

Kohl/2000-44

Run-Time Fault Monitor

- One Checkpointing Daemon (CPD) Per Host
 - ⇒ Ckpt Collector / Provider
 - ⇒ Run-Time Monitor
 - ⇒ Console for Restart / Migrate
- CPDs Comprise Fault-Tolerant Application...
 - ⇒ Handle Failure of Host / CPD
 - ⇒ Coordinate Redundancy
 - ⇒ Ring Topology



ORNL

Kohl/2000-45

CUMULVS Checkpoint Initialization

- C: `int cp_restart = stv_cpInit(char *aout_name,
int notify_tag, int *ntasks);`
- Fortran: `stvfcpinit(aout_name, notify_tag, ntasks,
cp_restart)`

where:

- `cp_restart` ~ status value, indicates restart / failure recovery
 - * use to circumvent normal application startup and data initialization!!
- `aout_name` ~ name of executable file to spawn for restart
- `notify_tag` ~ message code for notify messages (PVM only)
- `ntasks` ~ number of tasks in application
 - * in a restart condition, this value can be set by CUMULVS!

Note: `stv_cpInit()` must be called **after** `stv_init()`!

ORNL

Kohl/2000-46

Alternate Checkpoint Restart Check

- C: `int cp_restart = stv_isCpRestart();`
- Fortran: `stvfiscprestart(cp_restart)`

where:

→ `cp_restart` ~ status value, indicates restart / failure recovery
* use to circumvent normal application startup and data initialization!!

(Additional routine to check for Restart / Failure Recovery.)

ORNL

Kohl/2000-47

Checkpoint Collection

- Actually Collect Data from Local Task
- Invoke When Parallel Data / State Consistent
 - ⇒ Highly Non-Trivial in the General Case!!
 - ⇒ Straightforward for Most Iterative Applications
 - Save Checkpoint at Beginning or End of Main Loop

- C: `int status = stv_checkpoint();`
- Fortran: `stvfcheckpoint(status)`

where:

→ `status < 0` indicates system failure

Control Overhead Using Checkpointing Frequency:

```
if ( !(i % 100) ) stv_checkpoint();
```

ORNL

Kohl/2000-48

Restoring Data From A Checkpoint

- Once Restart Condition Has Been Identified
 - Can Be Invoked Incrementally to Bootstrap Data
 - ⇒ Interlace With Data Field / Parameter Definitions...
 - ⇒ Automatically Loads Data Back Into User Memory for All Defined Data Fields and Parameters
 - C: `int remaining = stv_loadFromCP();`
 - Fortran: `stvloadfromcp(remaining)`
- where:
- remaining ~ number of remaining data fields or parameters to define / restore

ORNL

Kohl/2000-49

Manual Software Instrumentation

- SPDT 98 Case Study ~ SW Instrumentation Cost

Instrumentation:	Seismic:	Wing Flow:
Original Lines of Code	20,632	2,250
Vis / Steer System Init	3	3
Vis / Steer Variable Decls	48	73
CP Restart Initialization	21	12
CP Rollback Handling	41	34
Total Instrumentation	204 ~ 1.0 %	188 ~ 7.7 %

ORNL

Kohl/2000-50

Checkpointing Efficiency

- SPDT 98 Case Study ~ Execution Overhead

Seconds per Iteration

Experiment:	SGI:	Cluster:	Hetero:
Seismic - No Checkpointing	2.83	6.23	9.46
Seismic - Checkpoint for Restart	2.99	6.50	10.76
Seismic - Checkpoint for Rollback	3.03	6.66	10.90
Wing - No Checkpointing	0.69	1.58	6.14
Wing - Checkpoint for Restart	0.77	1.71	7.10
Wing - Checkpoint for Rollback	0.79	1.71	7.30

(Checkpointing Every 20 Iters.)

Seismic Overhead: 4-14% Restart, +1-3% Rollback.

Wing Overhead: 8-15% Restart, +0-2.5% Rollback.

ORNL

Kohl/2000-51



PVM (Parallel Virtual Machine)

- Use Arbitrary Collection of Computers as a Single, Large, Uniform Parallel Computer
 - ⇒ Workstations, PCs (Unix or NT) ~ Clusters
 - ⇒ SMPs, MPPs
 - ⇒ Connected by a Network
- Programming Model & Runtime System
 - ⇒ Message-Passing ~ “Point-to-Point”, Context
 - ⇒ Process Control, Message Mailbox Database
 - ⇒ Fault Notification

ORNL

Kohl/2000-52

PVM vs. MPI: Different Goals

- MPI
 - ⇒ Stable Standard, Portable Code.
 - ⇒ High-Performance on Homogeneous Systems.
- PVM
 - ⇒ Research Tool, Robust, Interoperable.
 - ⇒ Good Performance on Heterogeneous Systems.

ORNL

Kohl/2000-53

PVM vs. MPI: Different Philosophies

- MPI
 - ⇒ Static Model (MPI_SPAWN in MPI-2...)
 - ⇒ “Rich” API (MPI-1 / 128, MPI-2 / 288)
 - ⇒ Performance
- PVM
 - ⇒ Dynamic Model
 - ⇒ “Simple” API (PVM 3.4 / 75)
 - ⇒ Flexibility

ORNL

Kohl/2000-54

Portability vs. Interoperability

- Portable:
 - ⇒ Re-compile Source Without Modification on a Different System.
 - ⇒ True of Both MPI and PVM.
- Interoperable:
 - ⇒ Executables on Different Systems Communicate
 - ⇒ Yes PVM, Sometimes MPI (Not Required)
 - ⇒ Different MPI Implementations? Soon ~ IMPI.

ORNL

Kohl/2000-55

Language Support

- Write Programs in C, Fortran, C++, F90?
 - ⇒ Yes, in Both MPI and PVM.
- Communicate Among Such Programs?
 - ⇒ Yes, in PVM.
 - ⇒ MPI? Maybe... Not Required by Standard.

ORNL

Kohl/2000-56

Comparison Scenario 1: Homogeneous Systems

- Interoperability is Irrelevant
- MPI
 - ⇒ Best Performance using Optimized Native Comm.
- PVM
 - ⇒ Trades Performance for Flexibility, Unnecessary.
- MPI “Wins”!

ORNL

Kohl/2000-57

Comparison Scenario 2: Heterogeneous Systems

- Common Situation
 - ⇒ Front-End / Back-End Model.
- PVM
 - ⇒ Transparent Handling by Default, “It Works.”
- MPI
 - ⇒ No Guarantees... Implementation Dependent
 - ⇒ No Vendor Cooperation? (Performance Loss...)
- PVM “Wins”!

ORNL

Kohl/2000-58

Performance vs. Flexibility

- To Be Flexible, You Must Pay the Price.
- Overheads:
 - ⇒ Data Conversion, Network Protocol Selection,
Extra Message Headers (on top of Native Comm)...
- Choose the Lowest Common Denominator.
 - ⇒ Not the Best on Any System.
- Performance Dictates Locally Optimal Solution.
 - ⇒ Lose Interoperability.

ORNL

Kohl/2000-59

Interesting Result

- Someone could build an MPI implementation that supports interoperability across different systems / languages (Mpich, LAM, IMPI...).
- But:
 - ⇒ It Would Perform About the Same as PVM!!

ORNL

Kohl/2000-60

Supporting MPI Applications in CUMULVS

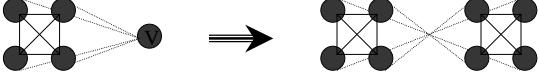
- Vendor-Supported Standard, Growing Application Base
⇒ Selected for ASCI Applications
- MPI Built for High Performance
 - ⇒ Static Model, Minimal Operating Environment
 - ⇒ No Name Service / Database, Fault Recovery / Notification?
 - ⇒ MPI_SPAWN()...? Proxy Server for Viewer Attachment?
- Existing CUMULVS Solution:
 - ⇒ Applications Can Communicate Using MPI
 - ⇒ CUMULVS Viewers / CPDs Still Attach Using PVM
- Possible “Reduced-Functionality” MPI Version...?
⇒ Currently Under Development

ORNL

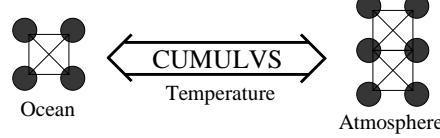
Kohl/2000-61

Future CUMULVS Plans (1 of 3)

Coupling Data Fields in Simulation Models

- Natural Extension to Viewer Scenario
⇒ Promote “Many-to-1” → “Many-to-Many”
- Translate Disparate Data Decompositions
⇒ Complements PAWS Coupling Work
⇒ Builds on CCA (Common Component Architecture) Forum

E.g. Regional Climate Assessment



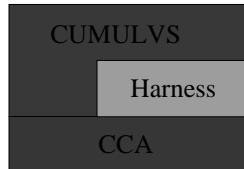
ORNL

Kohl/2000-62

Future CUMULVS Plans (2 of 3)

Building on Harness

- “Harness” ~ Next Generation HDC Environment
 - ⇒ Pluggable Virtual Machine, Distributed Control
 - Follow-on to PVM...
 - ⇒ Inspired by CUMULVS Coupling Needs
 - ⇒ ORNL, UT and Emory (Basic Research)
- Reinforces Need for CCA
 - ⇒ Harness Pluggability Builds on CCA Foundation



ORNL

Kohl/2000-63

Future CUMULVS Plans (3 of 3)

- Application Interface:
 - ⇒ Assist Manual Instrumentation of Applications
 - GUI, Pre-Compiler...
- Checkpointing Efficiency:
 - ⇒ Tasks Write Data in Parallel / Parallel File System?
 - ⇒ Redundancy Levels, Improve Scalability
- Portability:
 - ⇒ Other Messaging Substrates
 - Reduced Functionality for MPI

ORNL

Kohl/2000-64

CUMULVS Summary

- Interact with Scientific Simulations
 - ⇒ Dynamically Attach Multiple Visualization Front-Ends
 - ⇒ Steer Model & Algorithm Parameters On-The-Fly
 - ⇒ Automatic Heterogeneous Fault Recovery & Migration
- Future Opportunities
 - ⇒ Couple Disparate Simulation Models
 - ⇒ Integrate with Other Tools and Systems via CCA
 - ⇒ Application Instrumentation GUI / Pre-Compiler

<http://www.csm.ornl.gov/cs/cumulvs.html>

ORNL

Kohl/2000-65

Seismic Example ~ 2D (Tcl/Tk)

ORNL

Kohl/2000-66

Seismic Example ~ 3D (AVS)

ORNL

Kohl/2000-67

Air Flow Over Wing Example ~ 3D (AVS)

ORNL

Kohl/2000-68